DIGITAL SYSTEM DESIGN LABORATORY

LAB 5

SINGLE CYCLE MICROPROCESSOR DATAPATH DESIGN

I. LAB OBJECTIVES

This Lab experiments are intended to design and test a Single Cycle Microprocessor

II. DESCRIPTION

Single Cycle Microprocessor datapath to be implemented is in figure 2.1.

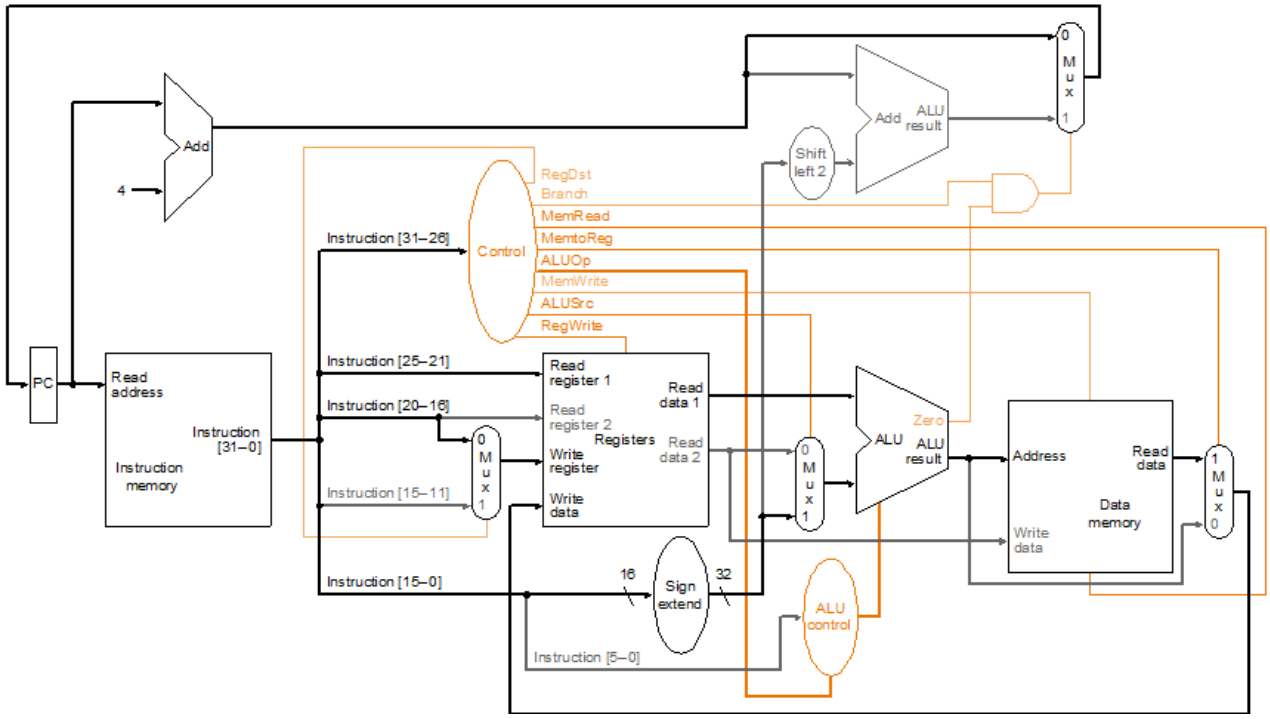


Figure 2.1: Single Cycle Microprocessor DataPath

III. LAB PROCEDURE

III.1 EXPERIMENT NO. 1

III.1.1 AIM: To understand and write the assembly code using MIPS Instruction set

register number of MIPS compiler conventions

Name	Register number	Usage
$zero	0	the constant value 0
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved (by callee)
$t8-$t9	24-25	more temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

MIPS Assembly Language Sumarize:

MIPS assembly language				
Category	Instruction	Example	Meaning	Comments
	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
Arithmetic	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`	Three operands; data in registers
	add immediate	`addi $s1, $s2, 100`	`$s1 = $s2 + 100`	Used to add constants
	load word	`lw $s1, 100($s2)`	`$s1 = Memory[$s2 + 100]`	Word from memory to register
	store word	`sw $s1, 100($s2)`	`Memory[$s2 + 100] = $s1`	Word from register to memory
Data transfer	load byte	`lb $s1, 100($s2)`	`$s1 = Memory[$s2 + 100]`	Byte from memory to register
	store byte	`sb $s1, 100($s2)`	`Memory[$s2 + 100] = $s1`	Byte from register to memory
	load upper immediate	`lui $s1, 100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
	branch on equal	`beq $s1, $s2, 25`	if (`$s1 == $s2`) go to PC + 4 + 100	Equal test; PC-relative branch
Conditional branch	branch on not equal	`bne $s1, $s2, 25`	if (`$s1 != $s2`) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1, $s2, $s3`	if (`$s2 < $s3`) `$s1` = 1; else `$s1` = 0	Compare less than; for beq, bne
	set less than immediate	`slti $s1, $s2, 100`	if (`$s2 < 100`) `$s1` = 1; else `$s1` = 0	Compare less than constant
	jump	`j 2500`	go to 10000	Jump to target address
Uncondi-	jump register	`jr $ra`	go to `$ra`	For switch, procedure return
tional jump	jump and link	`jal 2500`	`$ra` = PC + 4; go to 10000	For procedure call

Operation code (Op code) sumarize:

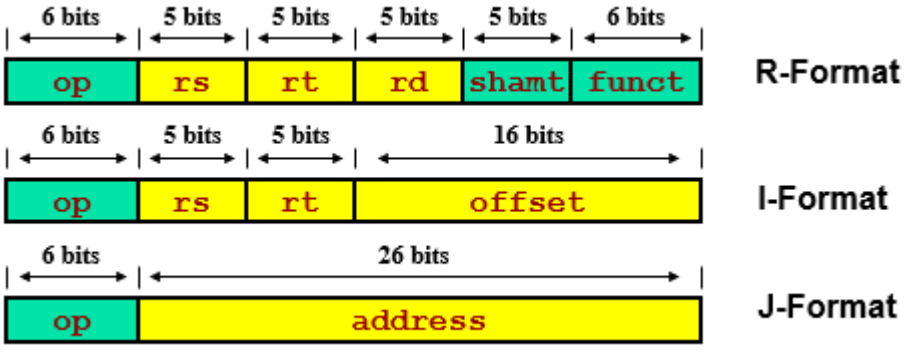
Op	Opcode name	Value
000000	R-format	Rformat1
000010	jmp	JUMP1
000100	beq	BEQ1
100011	lw	Mem1
101011	sw	Mem1

ALU opcode and Function

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at Funct
11	Not Used

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Add)
X1	X	110 (Subtract)
1X	100000 (add)	010 (Add)
1X	100010 (sub)	110 (Subtract)
1X	100100 (and)	000 (And)
1X	100101 (or)	001 (Or)
1X	101010 (slt)	111 (SLT)

Instruction Formats



III.1.2 CODE

a) Assembly Code sample 1:

Instruction	Meaning
addi $s0, $zero, 33	load immediate value 33 to register $s0
addi $s1, $zero, 66	load immediate value 66 to register $s1
add $s2, $s0, $s1	$s2 = $s0 + $s1
sub $s3, $s1, $s0	$s1 = $s1 – $s0
sw $s3, 10($s2)	Memory[$s2+10] = $s3
lw $s1, 10($s2)	$s1 = Memory[$s2+10]

b) Assembly Code sample 2:

Assume the code start from address PC=0x00000000, one instruction is store in one memory location.

Instruction	Meaning
addi $s2, $zero, 55	load immediate value 55 to register $S2
addi $s3, $zero, 22	load immediate value 22 to register $S3
addi $s5, $zero, 33	load immediate value 55 to register $S3
add $s4,$s2,$s3	$s4 = $s2 + $s3
sub $s1,$s2,$s3	$s1 = $s2 – $s3
sw $s1,100($s2)	Memory[$s2+100] = $s1
lw $s1,100($s2)	$s1 = Memory[$s2+100]
bne $s1,$s5,End	Next instr. is at End if $s4 !=$s5
addi $s6, $zero, 10	load immediate value 10 to register $s6
beq $s4,$s5, End	Next instr. is at End if $s4 = $s5
addi $s6, $zero, 20	load immediate value 20 to register $s6
End: j End	jump Here

III.1.3 LAB ASSIGNMENT

a) Compile the Assembly **Assembly Code sample 1** into machine code (decimal code and binary code)
b) Explain briefly the meaning of **Assembly Code sample 1**

c) Compile the Assembly **Assembly Code sample 2** into machine code (decimal code and binary code)
d) Explain briefly the meaning of **Assembly Code sample 2**

III.2 EXPERIMENT NO. 2

III.2.1 AIM: To implement R-Type Datapath

Name	Register number	Usage
$zero	0	the constant value 0
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved (by callee)
$t8-$t9	24-25	more temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Given the assembly code

addi $s1, $zero, 33 load immediate value 33 to register $s0
addi $s2, $zero, 66 load immediate value 66 to register $s1
add $s0, $s1, $s2 **$s0 = $s1 + $s2**

Translate into Macchine code:

Assembly Code

```
add $s0, $s1, $s2
```

Field Values

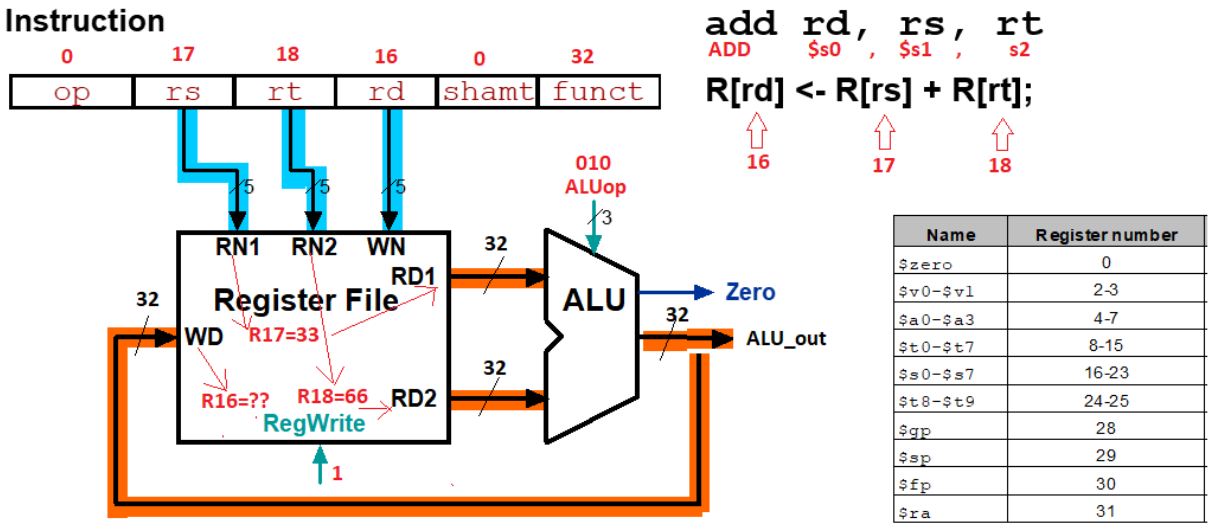
op	rs	rt	rd	shamt	funct
0	17	18	16	0	32

Machine Code

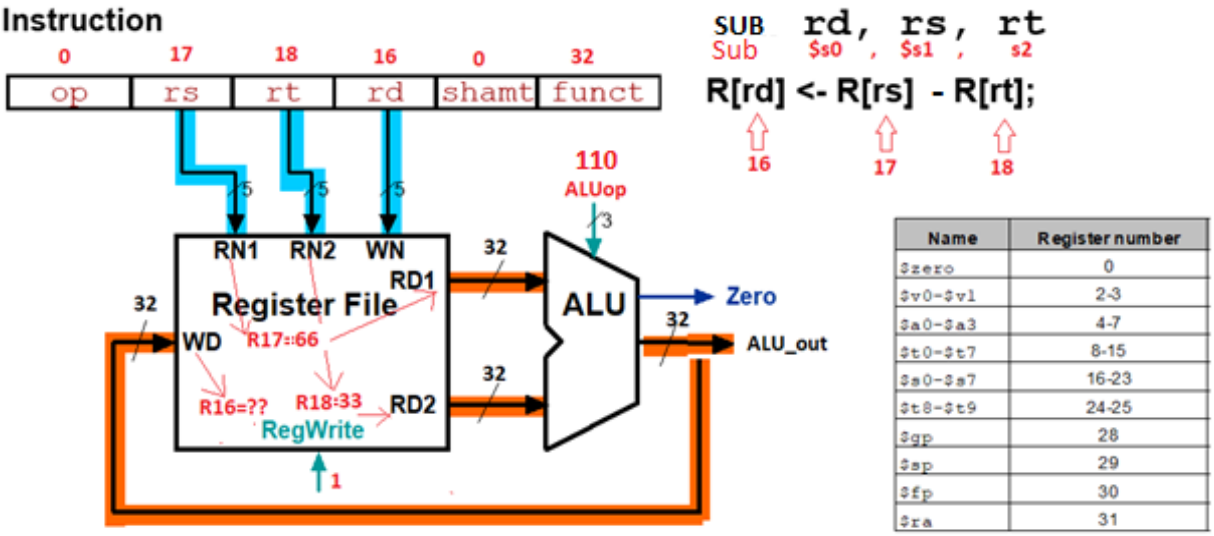
op	rs	rt	rd	shamt	funct
000000	10001	10010	10000	00000	100000

Modify the code register file:

- Assign initial value of the register 17=33 ($s1=33)
- Assign initial value of the register 18=33 ($s2=66)



Name	Register number
$zero	0
$v0-$v1	2-3
$a0-$a3	4-7
$t0-$t7	8-15
$s0-$s7	16-23
$t8-$t9	24-25
$gp	28
$sp	29
$fp	30
$ra	31



Name	Register number
$zero	0
$v0-$v1	2-3
$a0-$a3	4-7
$t0-$t7	8-15
$s0-$s7	16-23
$t8-$t9	24-25
$gp	28
$sp	29
$fp	30
$ra	31

III.2.2 CODE

```
module Datapath R_Type_Add(rs, rt, rd,ALUop,Zero,ALU_out);
// Your code here
endmodule
module Datapath R_Type_Sub(rs, rt, rd,ALUop,Zero,ALU_out);
// Your code here
endmodule
```

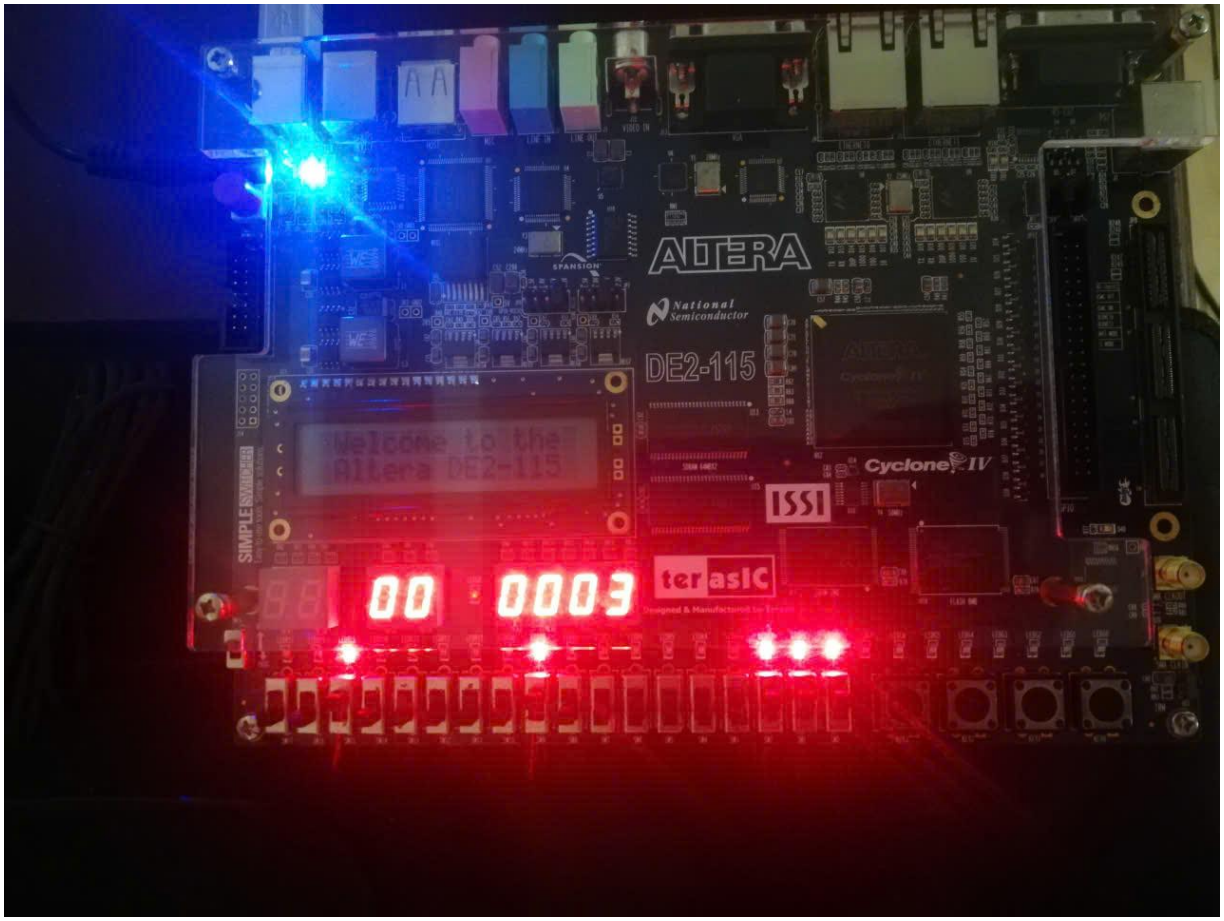
III.2.3 LAB ASSIGNMENT

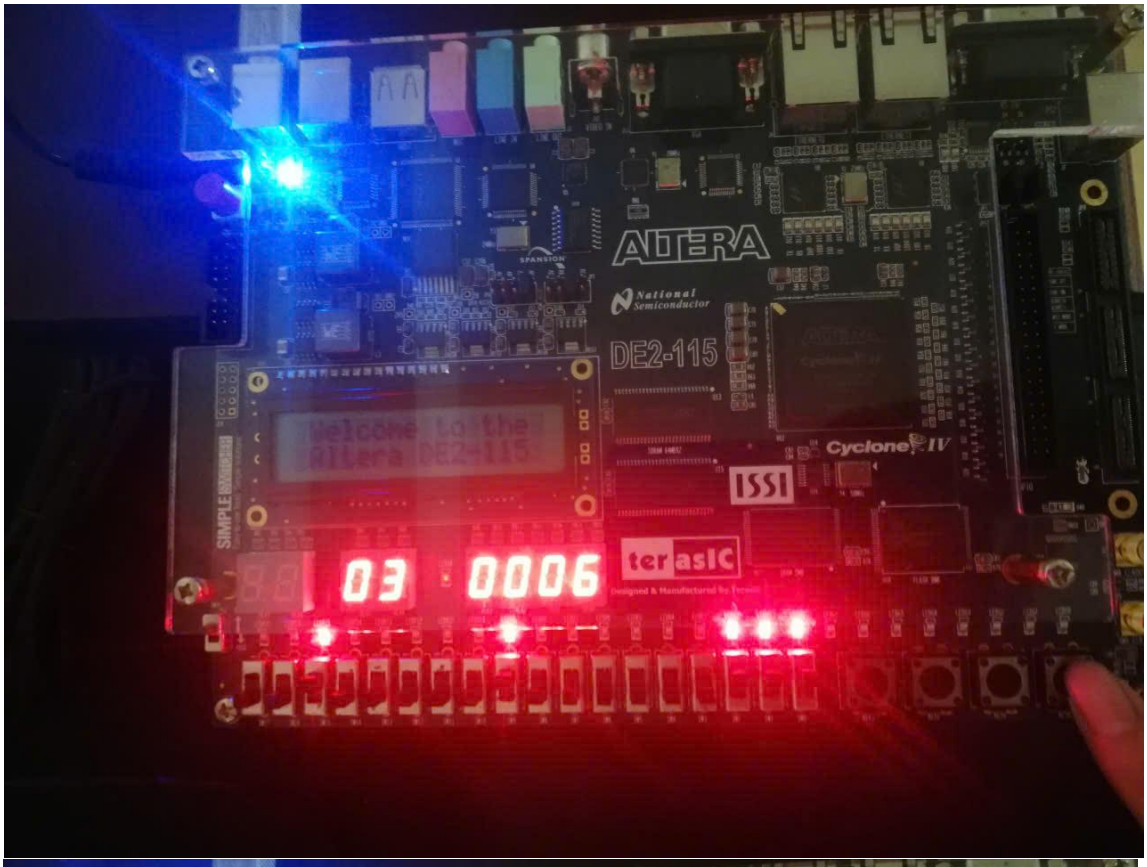
1) Write Verilog code to implement R_Type_Add module
2) Write testbenches to verify R_Type_Add module, simulate and check the simulation output data.
3) Write Verilog code to implement R_Type_Sub module

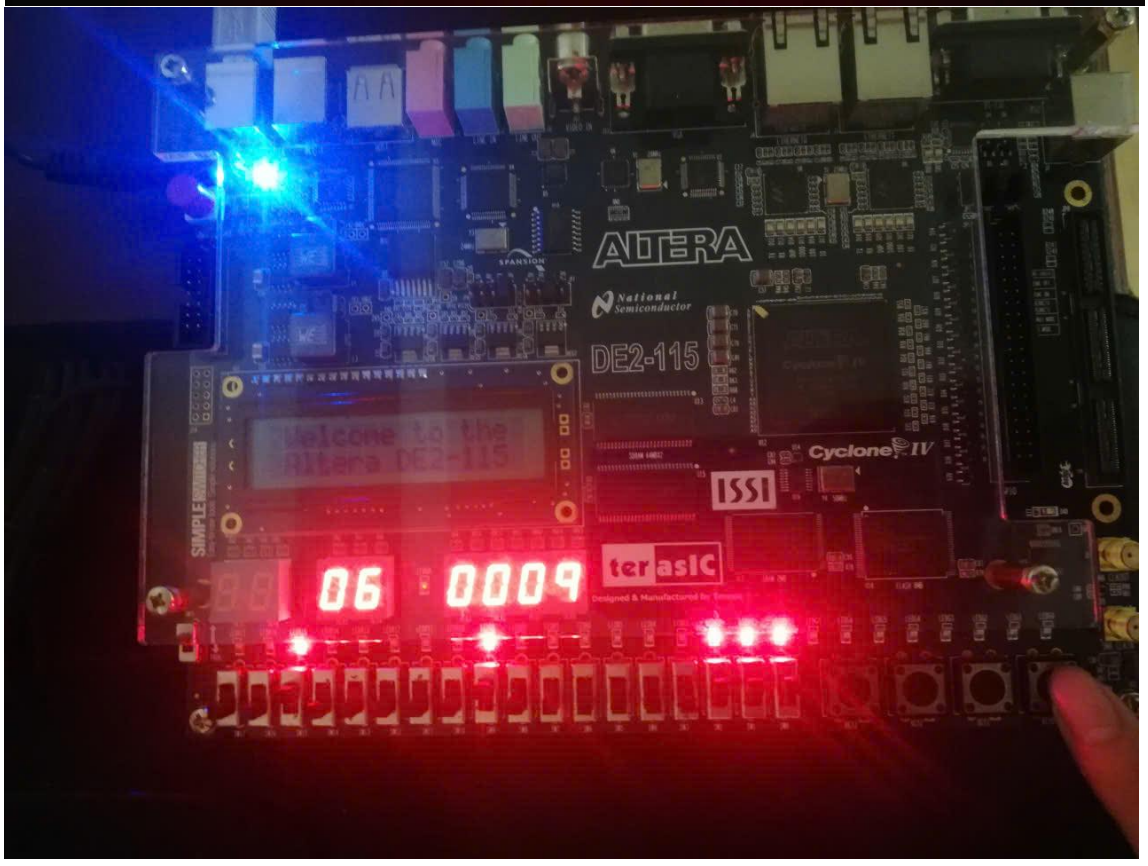
4) Write testbenches to verify R_Type_Sub , simulate and check the simulation output data.

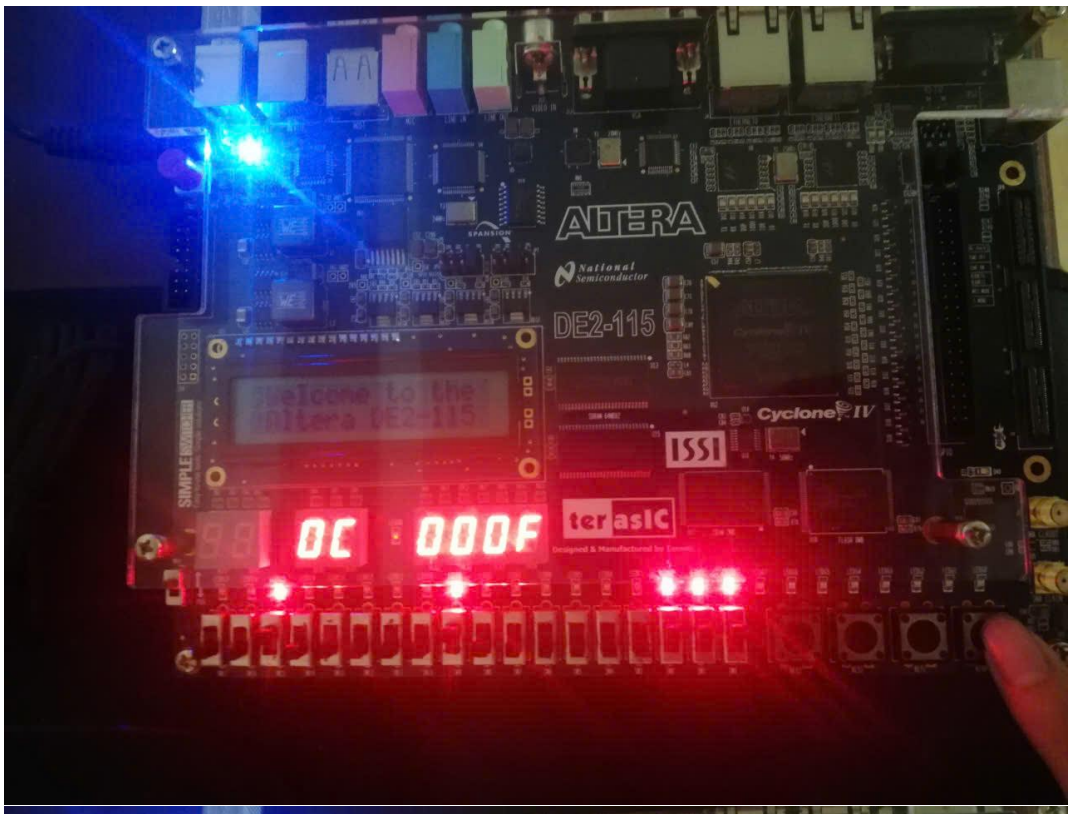
```
module lab5_ex2_v2(KEY, SW, LEDG, LEDR, HEX0, HEX1, HEX2, HEX3,
HEX4, HEX5);
   input [3:0] KEY;
   input [17:0] SW;
   output [7:0] LEDG;
   output [17:0] LEDR;
   output [0:6] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;

   wire [7:0] w_rd1, w_rd2, w_alu_out;

   assign LEDR = SW; // Đèn LED đỏ hiển thị trạng thái Switch

   // C1: Register File
   // Thay đổi: Dùng SW[0] làm RegWrite, KEY[0] làm Clock, KEY[1] làm Reset
   Register_File C1(
      .read_addr_1(SW[17:15]),
      .read_addr_2(SW[14:12]),
      .write_addr(SW[11:9]),
      .read_data_1(w_rd1),
      .read_data_2(w_rd2),
      .write_data(w_alu_out),
      .RegWrite(SW[0]),
      .clk(KEY[0]),
      .reset(KEY[1])
   );

   // C2: ALU
   // Thay đổi: Dùng SW[5:3] để chọn phép toán (tránh trùng với RegWrite)
   // Tìm dòng gọi module ALU, sửa rb_or_imm:
alu C2(
   .alufn(SW[5:3]),
   .ra(w_rd1),
   .rb_or_imm(SW[8:1]), // Lấy trực tiếp từ Switch 8 đến Switch 1 thay vì lấy từ
RegFile
   .aluout(w_alu_out),
   .zero(LEDG[7])
);
```

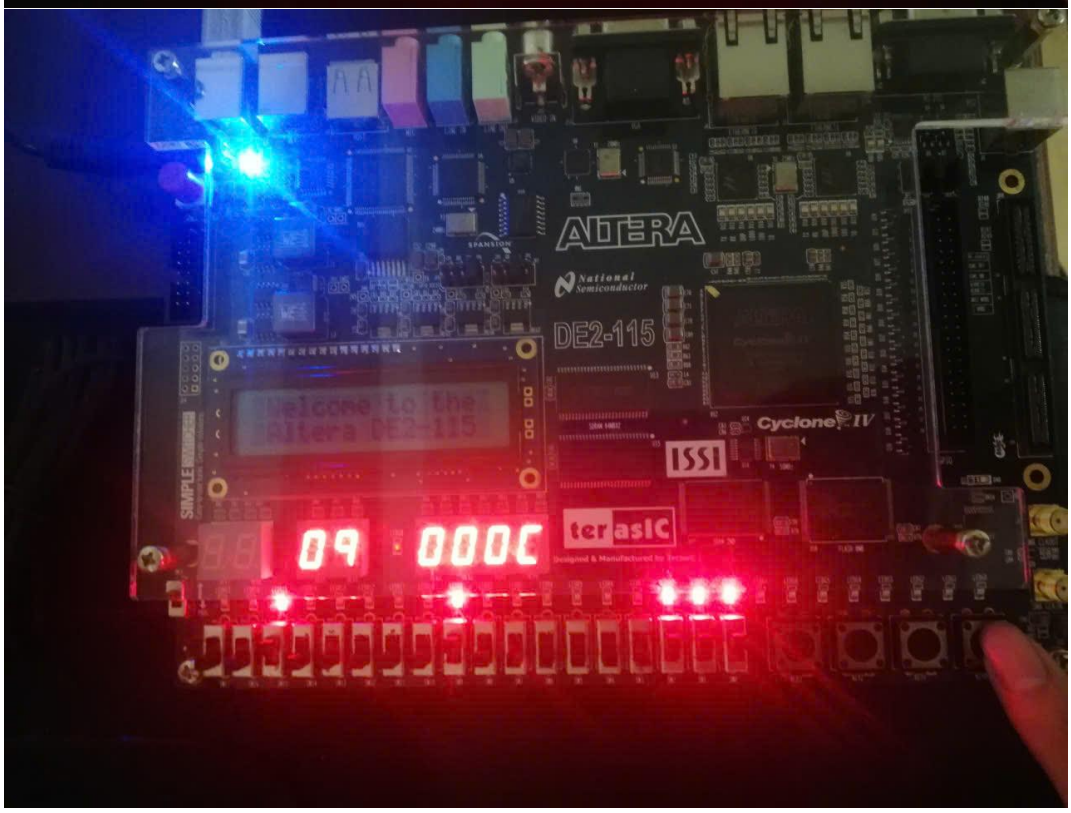
```
    // Hiển thị kết quả ALU lên HEX0, HEX1
    HEX_7SEG_DECODE H0(.BIN(w_alu_out[3:0]), .SSD(HEX0));
    HEX_7SEG_DECODE H1(.BIN(w_alu_out[7:4]), .SSD(HEX1));

    // Hiển thị giá trị đang đọc từ RegFile (Source B) lên HEX2, HEX3
    HEX_7SEG_DECODE H2(.BIN(w_rd2[3:0]), .SSD(HEX2));
    HEX_7SEG_DECODE H3(.BIN(w_rd2[7:4]), .SSD(HEX3));

    // Hiển thị giá trị đang đọc từ RegFile (Source A) lên HEX4, HEX5
    HEX_7SEG_DECODE H4(.BIN(w_rd1[3:0]), .SSD(HEX4));
    HEX_7SEG_DECODE H5(.BIN(w_rd1[7:4]), .SSD(HEX5));
endmodule
```

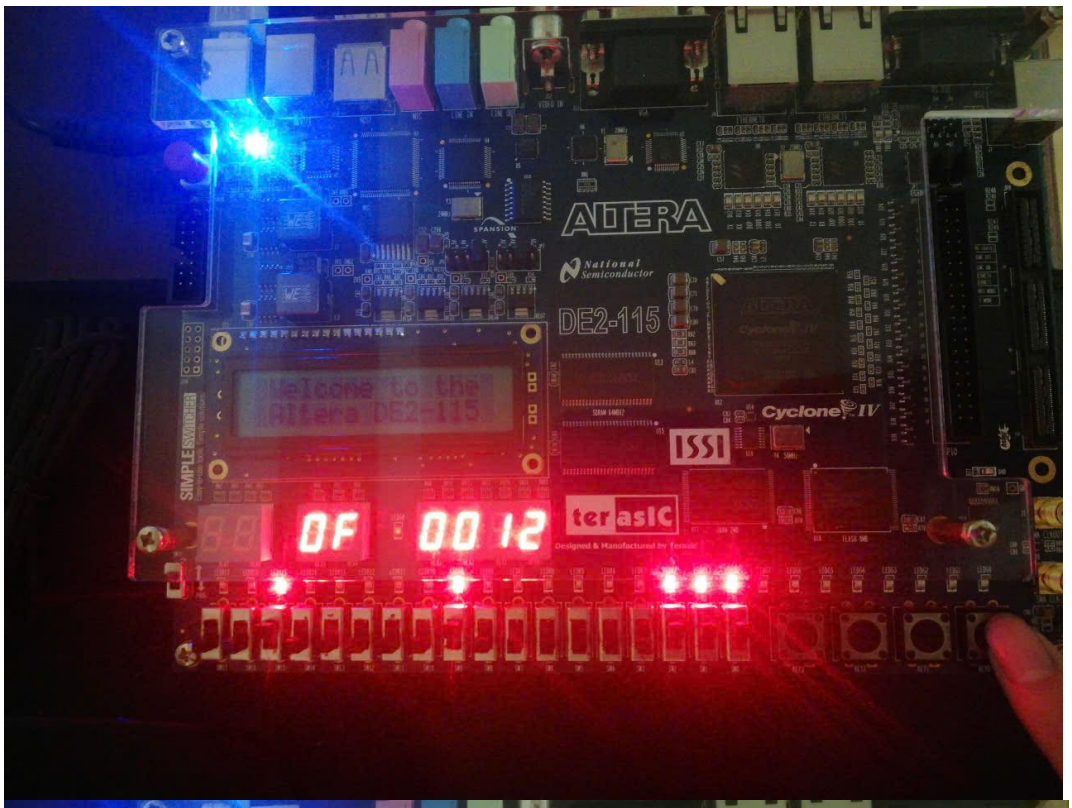


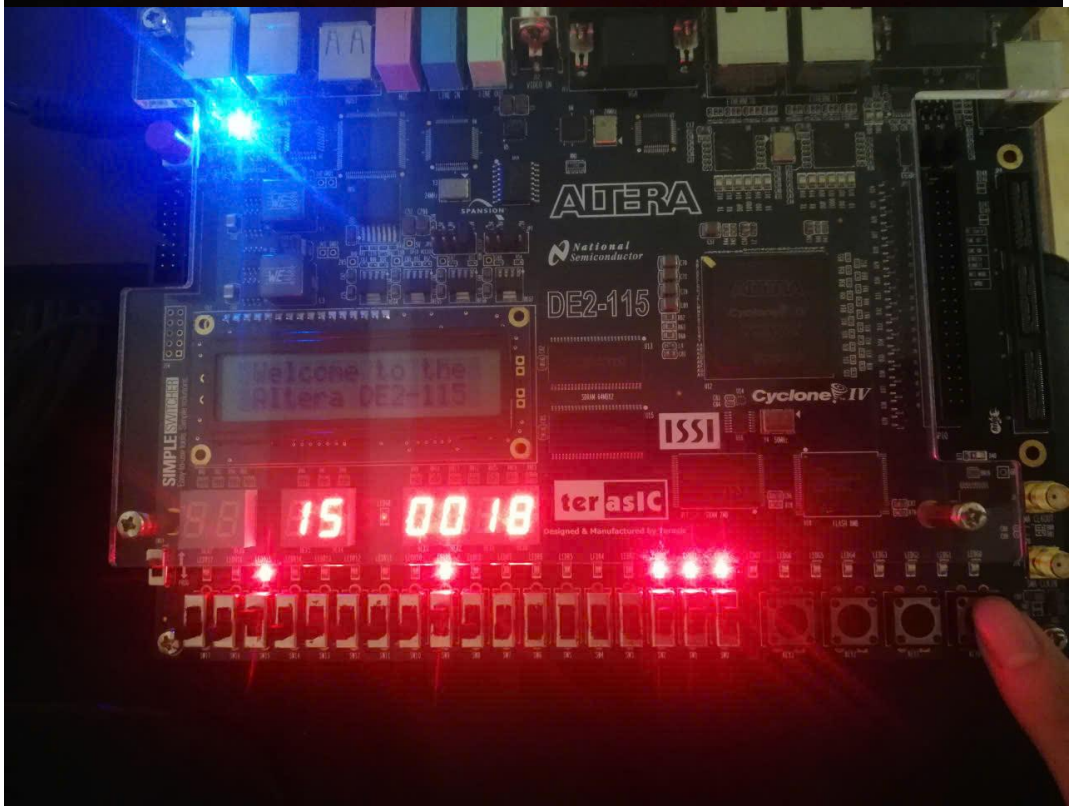

ALTERA
National Semiconductor
DE2-115
Cyclone IV
ISSI
terasIC
03 0006

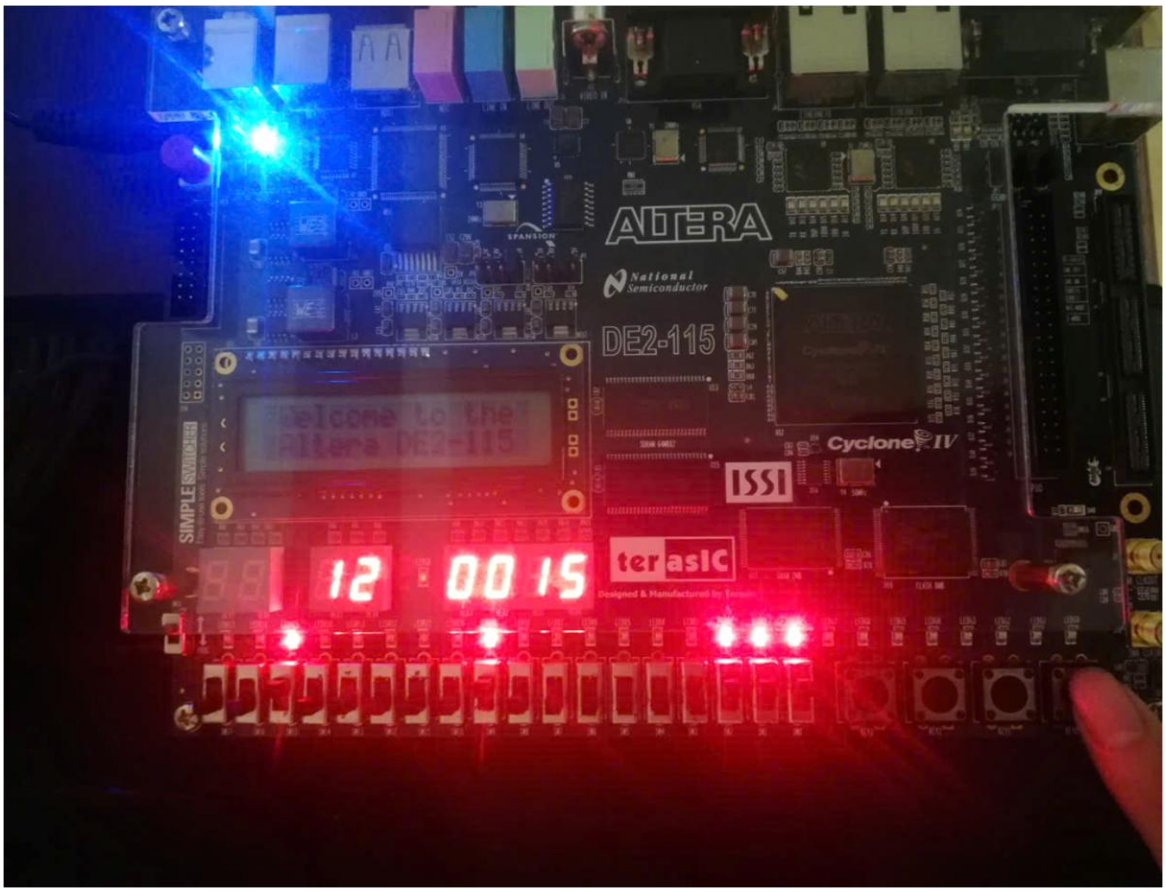

ALTERA
National Semiconductor
DE2-115
Cyclone IV
ISSI
terasIC
06 0009


ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
0C 000F


ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
09 000C

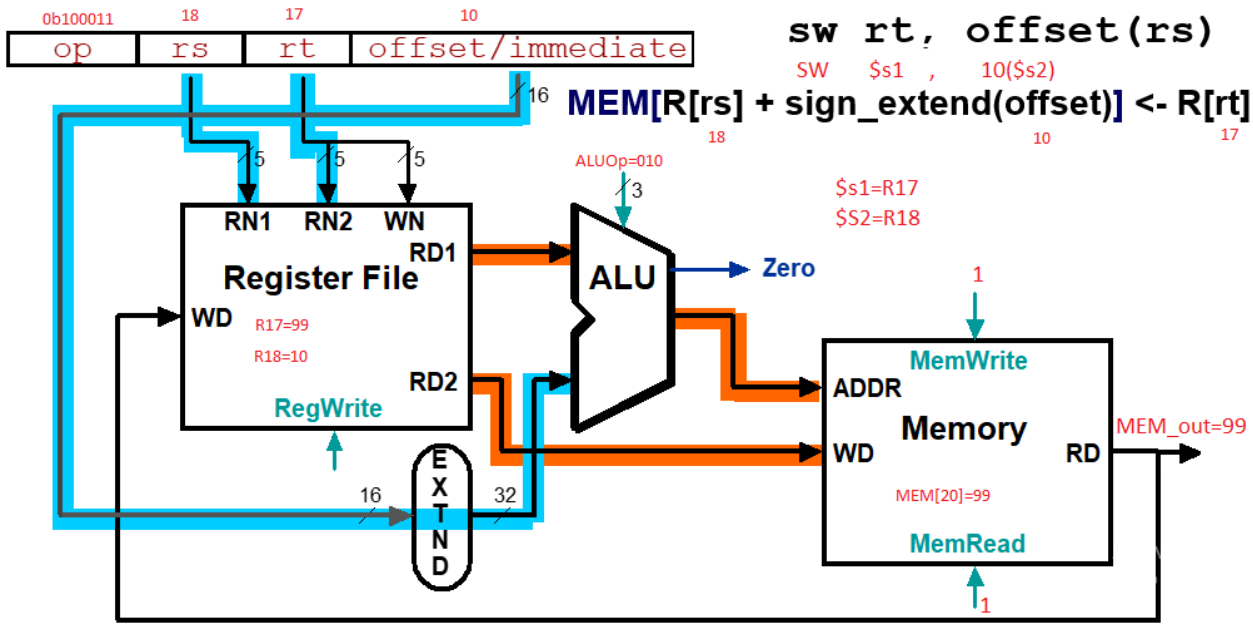

ALTERA
National Semiconductor
DE2-115
Cyclone IV
ISSI
terasIC
Welcome to the
Altera DE2-115
0F 00 12


ALTERA
National Semiconductor
DE2-115
Cyclone IV
ISSI
terasIC
Welcome to the
Altera DE2-115
15 00 18



III.3 EXPERIMENT NO. 3

III.3.1 AIM: To implement SW I-Type Instruction Datapath



III.3.2 CODE

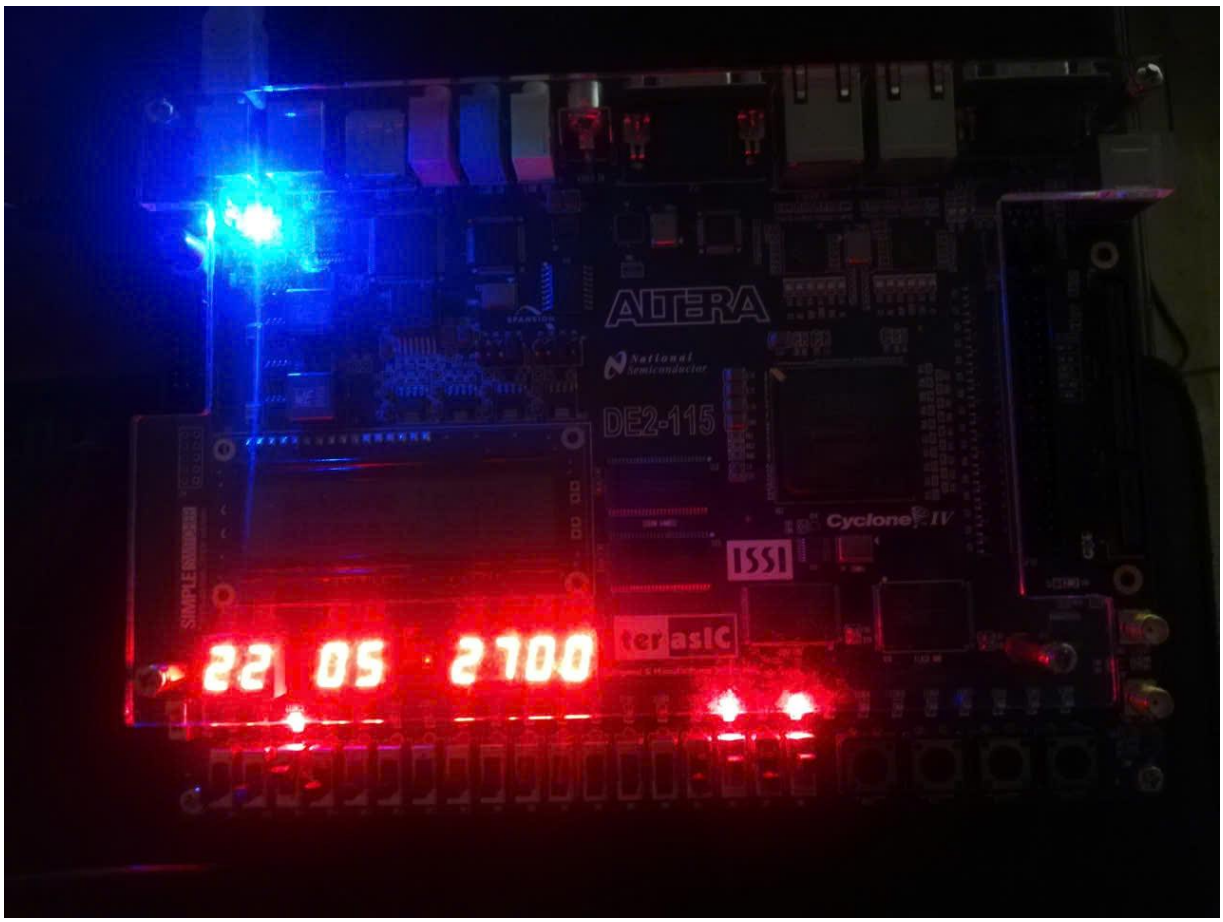
```
module SW_datapath (rs, rt, offset,ALUOp,MemWrite,MemRead,Mem_out);

    //Your Verilog code here

endmodule
```

III.3.3 LAB ASSIGNMENT

1) Write Verilog code to implement SW_datapath module
2) Write testbenches to verify SW_datapath module, simulate and verify the output data.

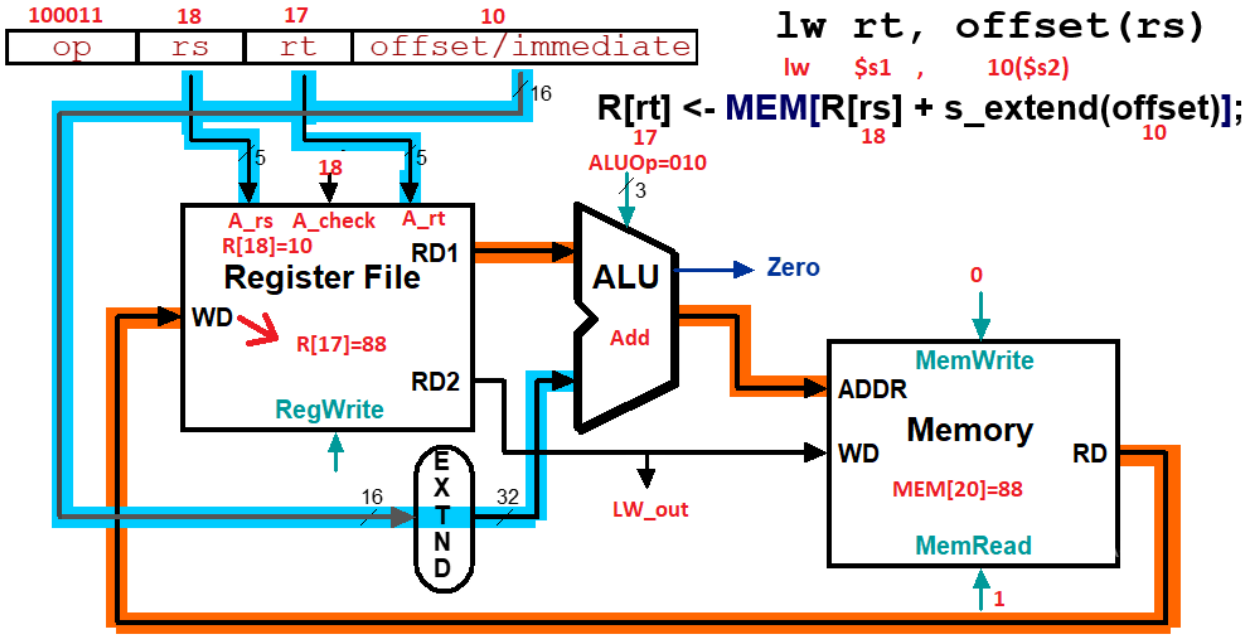
```
module lab5_ex3(
   input  [3:0]  KEY,
   input  [17:0] SW,
   output [7:0]  LEDG,
   output [17:0] LEDR,
   output [0:6]  HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7
);
   assign LEDR = SW;

   wire [7:0] W_rd1, W_rd2, W_alu_out, w_offset_ext, MEM_out;

   // Register file
   Register_File C1(
      .read_addr_1(SW[17:15]),
      .read_addr_2(SW[14:12]),
      .write_addr(3'b000),
      .read_data_1(W_rd1),
      .read_data_2(W_rd2),
      .write_data(MEM_out),
      .RegWrite(SW[11]),
      .clk(KEY[0]),
      .reset(KEY[1])
   );

   // ALU
   alu C2(
      .alufn(SW[10:8]),
      .ra(W_rd1),
      .rb_or_imm(w_offset_ext),
      .aluout(W_alu_out),
```

```
        .zero(LEDG[0])
    );

    // Sign extension
    Sign_Extension C3(
        .sign_in(SW[4:0]),
        .sign_out(w_offset_ext)
    );

    // Data memory
    Data_Memory C4(
        .addr(W_alu_out),
        .write_data(W_rd2),
        .read_data(MEM_out),
        .clk(KEY[0]),
        .reset(KEY[1]),
        .MemRead(SW[6]),
        .MemWrite(SW[5])
    );

    // HEX displays
    HEX_7SEG_DECODE h6 (.BIN(W_rd1[3:0]),      .SSD(HEX6));
    HEX_7SEG_DECODE h7 (.BIN(W_rd1[7:4]),      .SSD(HEX7));

    HEX_7SEG_DECODE h4 (.BIN(w_offset_ext[3:0]), .SSD(HEX4));
    HEX_7SEG_DECODE h5 (.BIN(w_offset_ext[7:4]), .SSD(HEX5));

    HEX_7SEG_DECODE h2 (.BIN(W_alu_out[3:0]),  .SSD(HEX2));
    HEX_7SEG_DECODE h3 (.BIN(W_alu_out[7:4]),  .SSD(HEX3));

    HEX_7SEG_DECODE h0 (.BIN(MEM_out[3:0]),    .SSD(HEX0));
    HEX_7SEG_DECODE h1 (.BIN(MEM_out[7:4]),    .SSD(HEX1));

endmodule
```


ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
22 05 2100

II.4 EXPERIMENT NO. 4

III.4.1 AIM: To implement LW I-Type Instruction Datapath



III.4.2 CODE

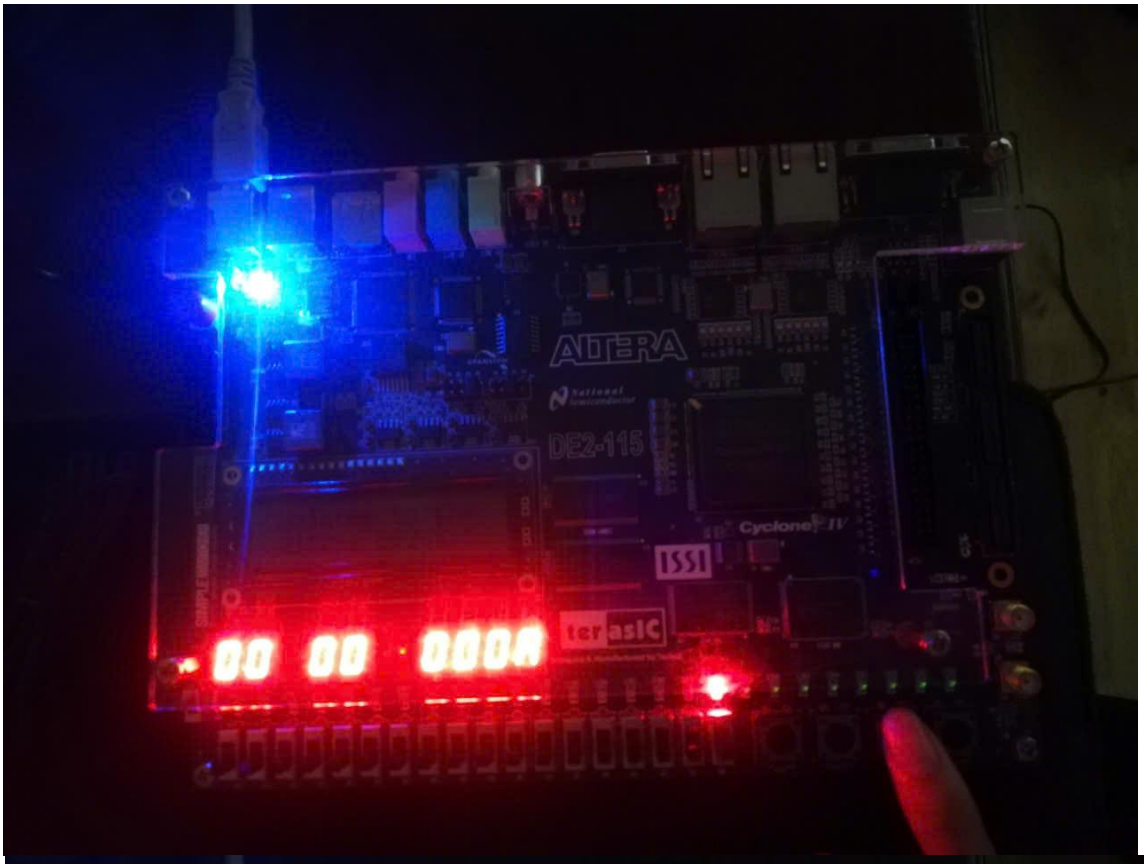
```
module LW_datapath (A_rs, A_rt, A_check, offset, ALUOp,MemWrite,MemRead,LW_out);

    //Your Verilog code here


endmodule
```

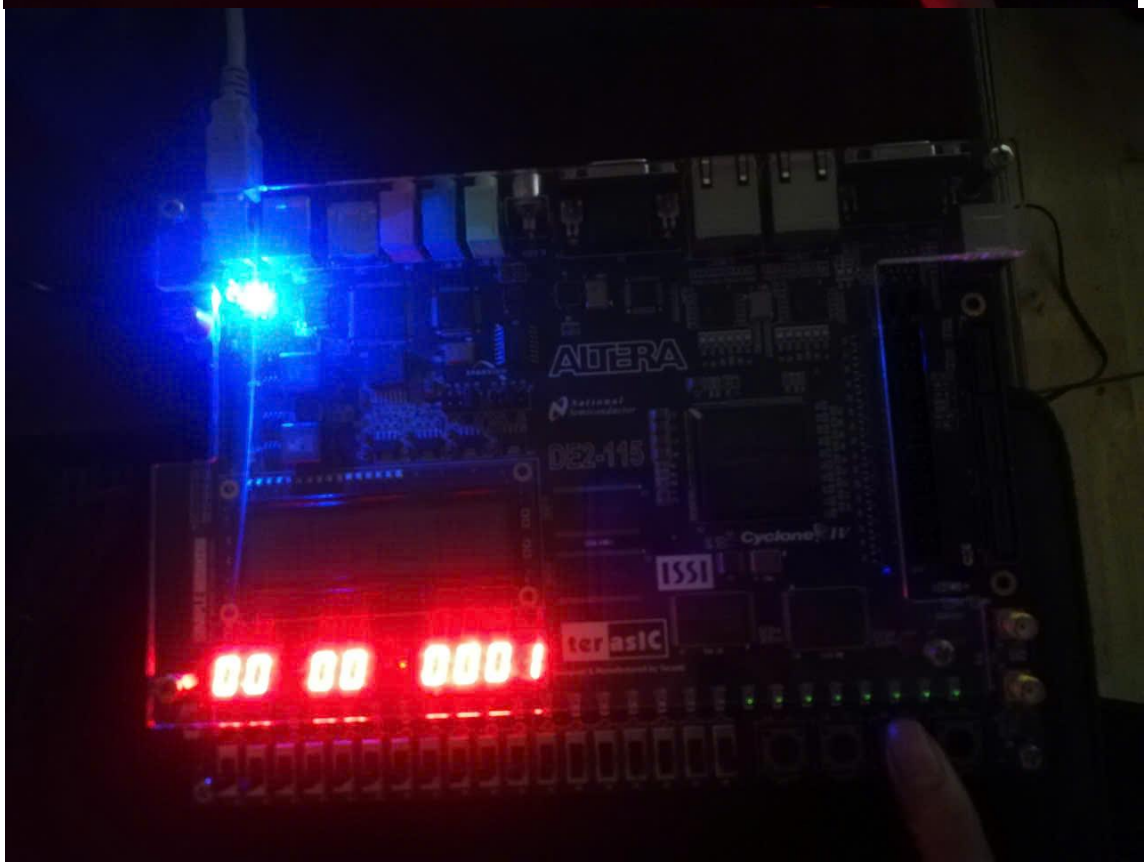
III.4.3 LAB ASSIGNMENT

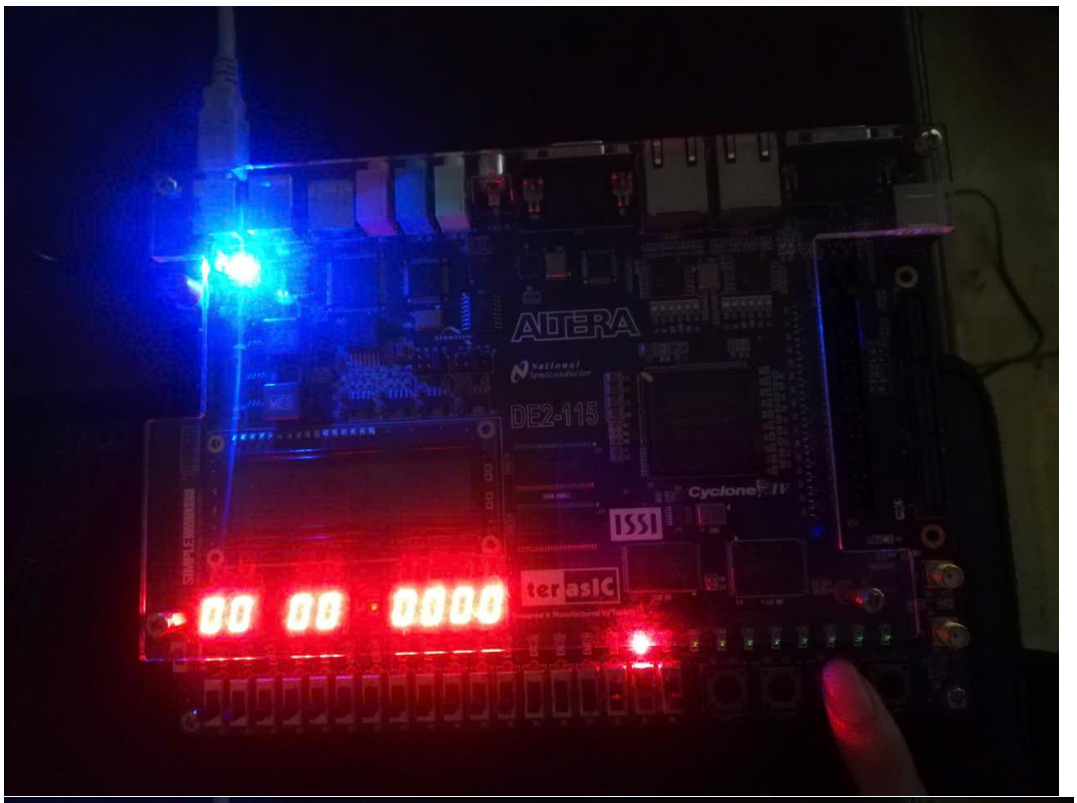
1) Write Verilog code to implement LW_datapath module
2) Write testbenches to verify LW_datapath module, simulate and verify the output data.

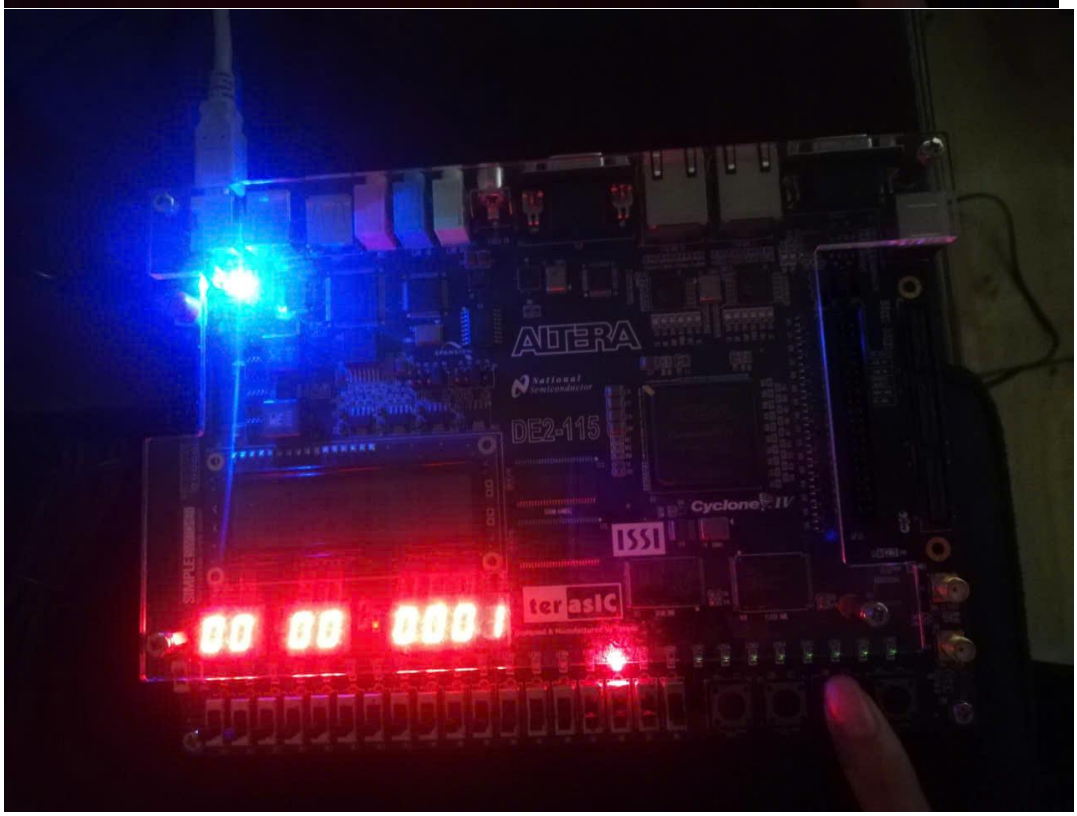
```
module lab5_ex4(
   input CLOCK_50,        // 50 MHz (không dùng nữa cho CPU nhưng vẫn giữ ở port)
   input [1:0] KEY,       // KEY[0] = reset, KEY[1] = Manual Clock
   input [9:0] SW,        // Switches
   output [31:0] LEDR,    // LEDs
   output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7
);
   // THAY ĐỔI TẠI ĐÂY:
```

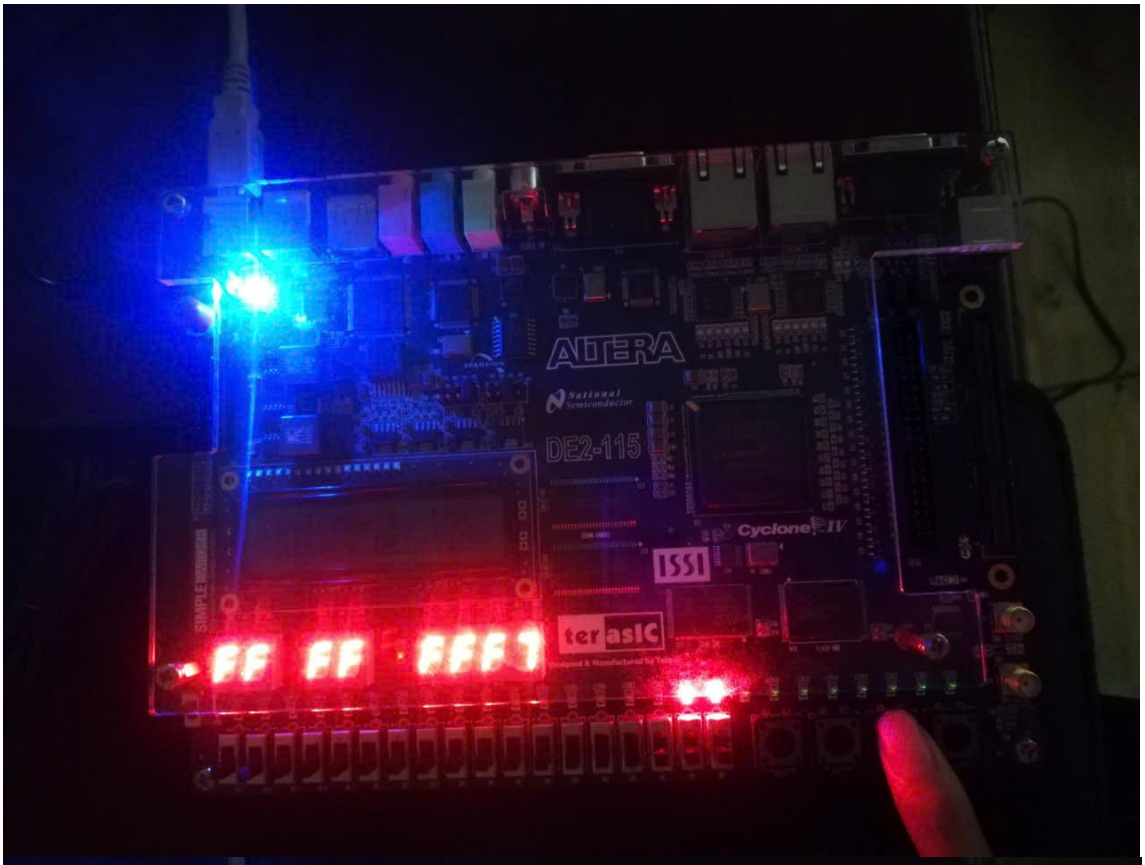
```
    // Đảo KEY[1] để khi NHẤN nút (0) sẽ tạo ra cạnh lên (posedge) cho CPU
    wire clk = ~KEY[1];
    wire reset = ~KEY[0];    // KEY[0] nhấn xuống = reset hệ thống

    // Processor signals
    wire [31:0] W_PC_out, instruction, W_RD1, W_RD2, W_m1, W_m2,
W_ALUout, W_MemtoReg;
    wire Jump;

    // Các phần còn lại giữ nguyên...
    single_c_Proc CPU(
        .reset(reset),
        .clk(clk),
        .W_PC_out(W_PC_out),
        .instruction(instruction),
        .W_RD1(W_RD1),
        .W_RD2(W_RD2),
        .W_m1(W_m1),
        .W_m2(W_m2),
        .W_ALUout(W_ALUout),
        .W_MemtoReg(W_MemtoReg),
        .Jump(Jump)
    );

    // SW[0] = 0: LED hiện PC | SW[0] = 1: LED hiện ALU Out
    assign LEDR = (SW[0] == 1'b0) ? W_PC_out : W_ALUout;

    // HEX displays (Giữ nguyên phần hiển thị ALUout)
    hex_display u0(.in(W_ALUout[3:0]),   .hex(HEX0));
    hex_display u1(.in(W_ALUout[7:4]),   .hex(HEX1));
    hex_display u2(.in(W_ALUout[11:8]),  .hex(HEX2));
    hex_display u3(.in(W_ALUout[15:12]), .hex(HEX3));
    hex_display u4(.in(W_ALUout[19:16]), .hex(HEX4));
    hex_display u5(.in(W_ALUout[23:20]), .hex(HEX5));
    hex_display u6(.in(W_ALUout[27:24]), .hex(HEX6));
    hex_display u7(.in(W_ALUout[31:28]), .hex(HEX7));

endmodule
```

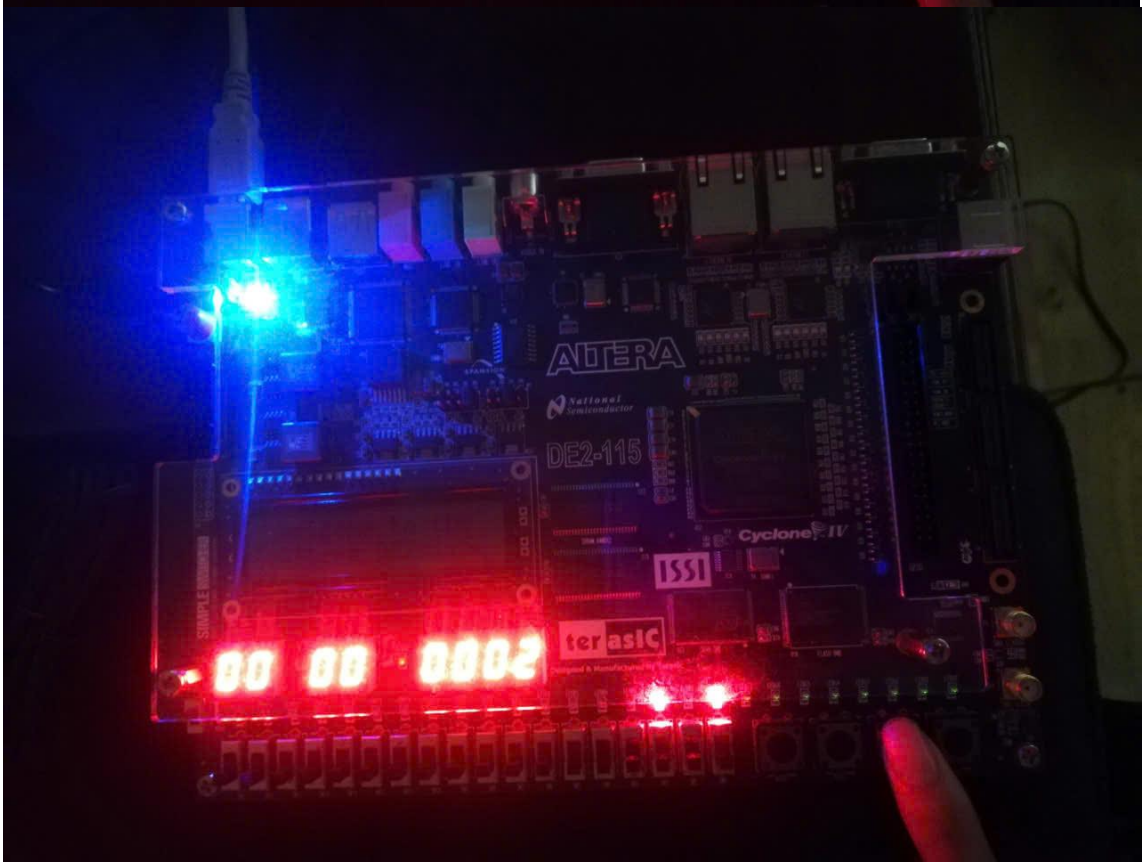

ALTERA
DE2-115
ISSI
terasIC

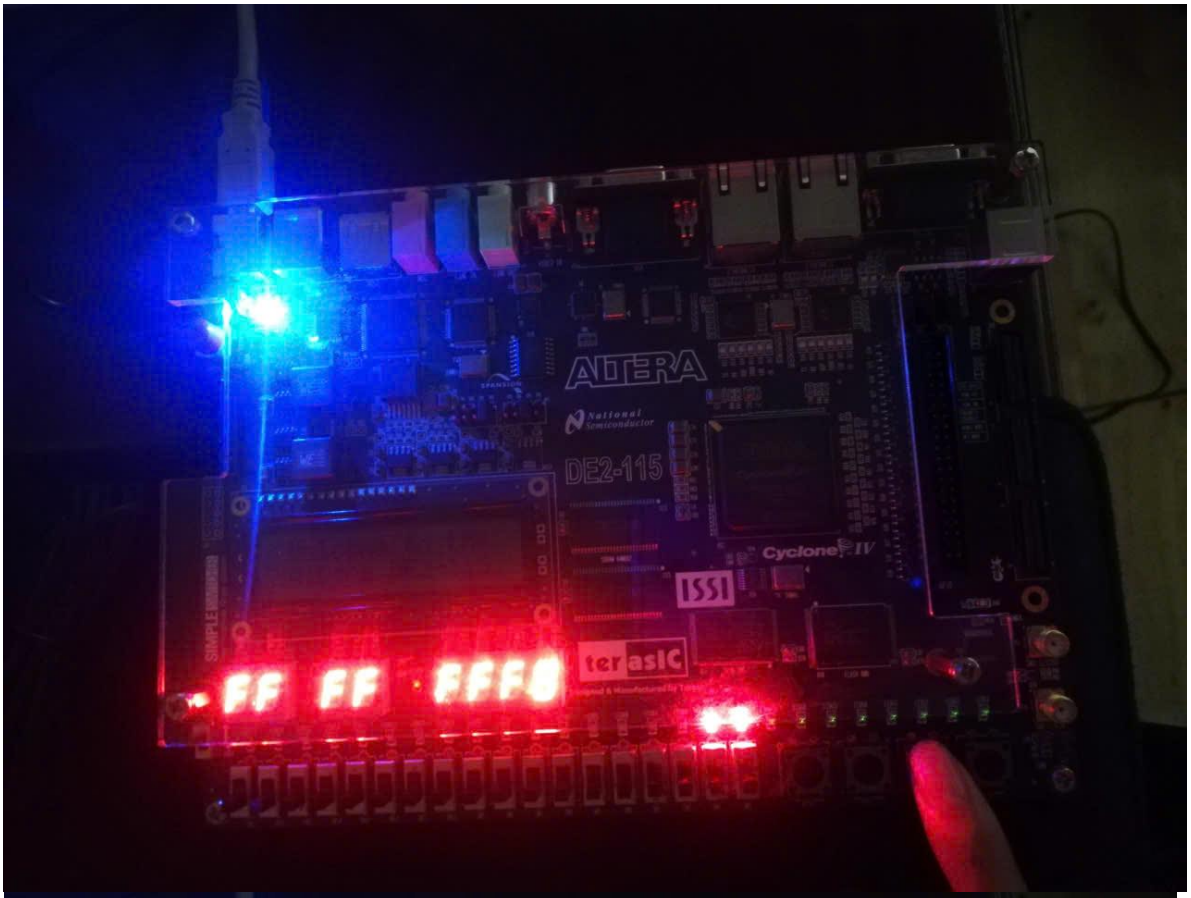

ALTERA
DE2-115
ISSI
terasIC

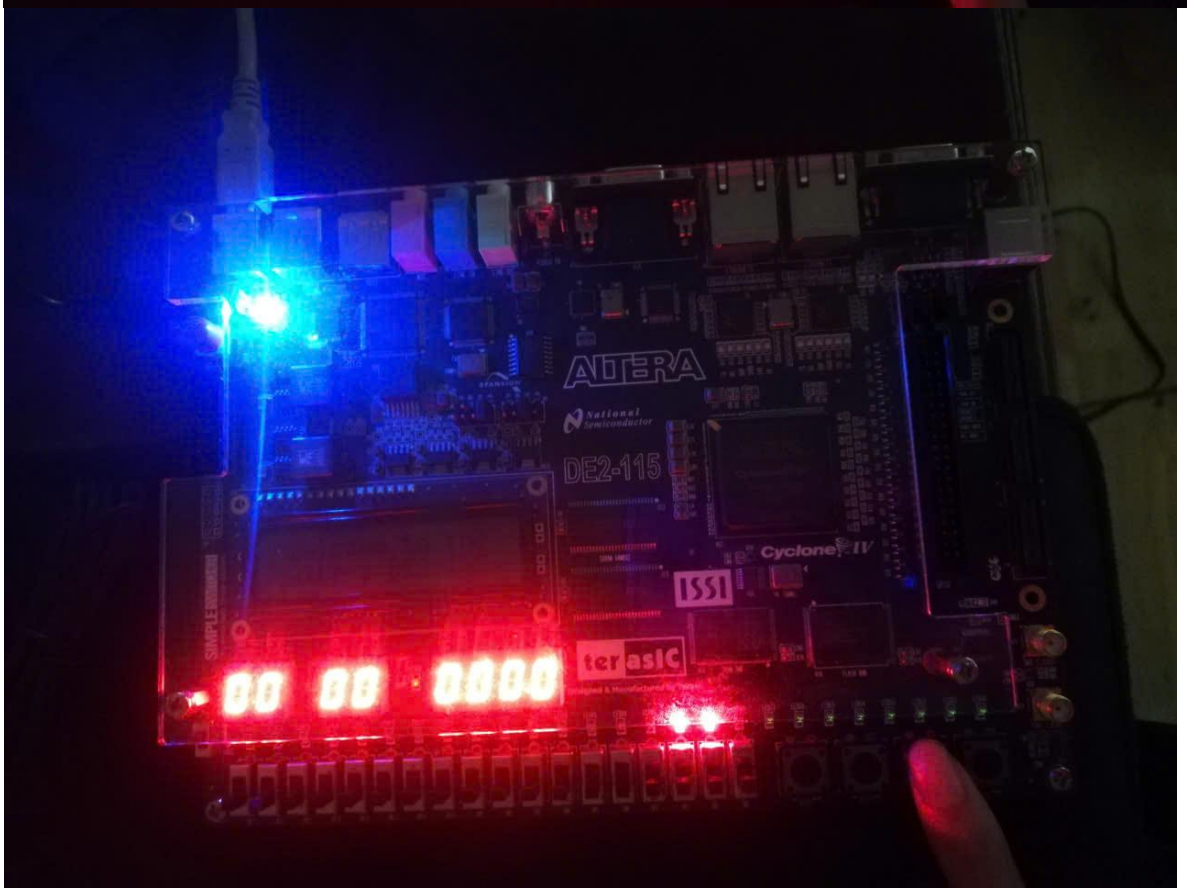

ALTERA
DE2-115
ISSI
terasIC

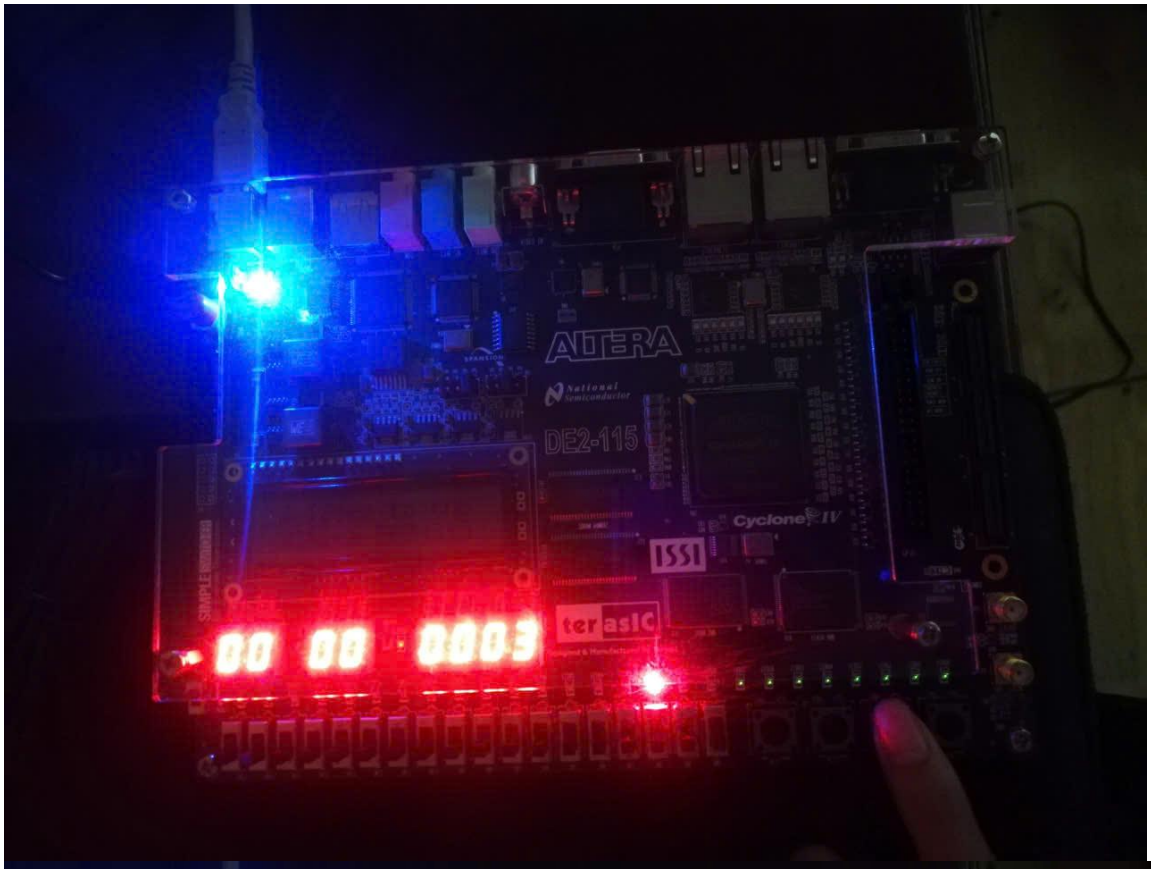

ALTERA
DE2-115
ISSI
terasIC

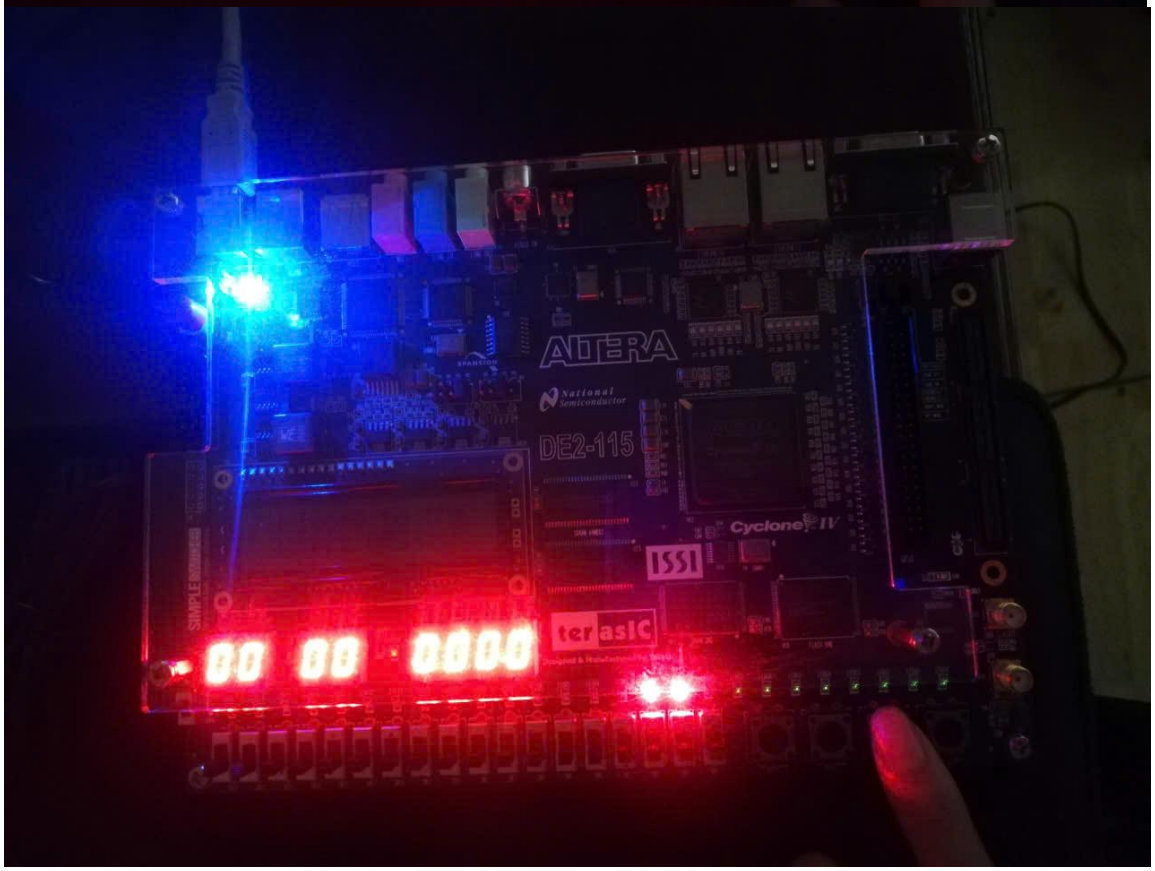

ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
FF FF FFF1

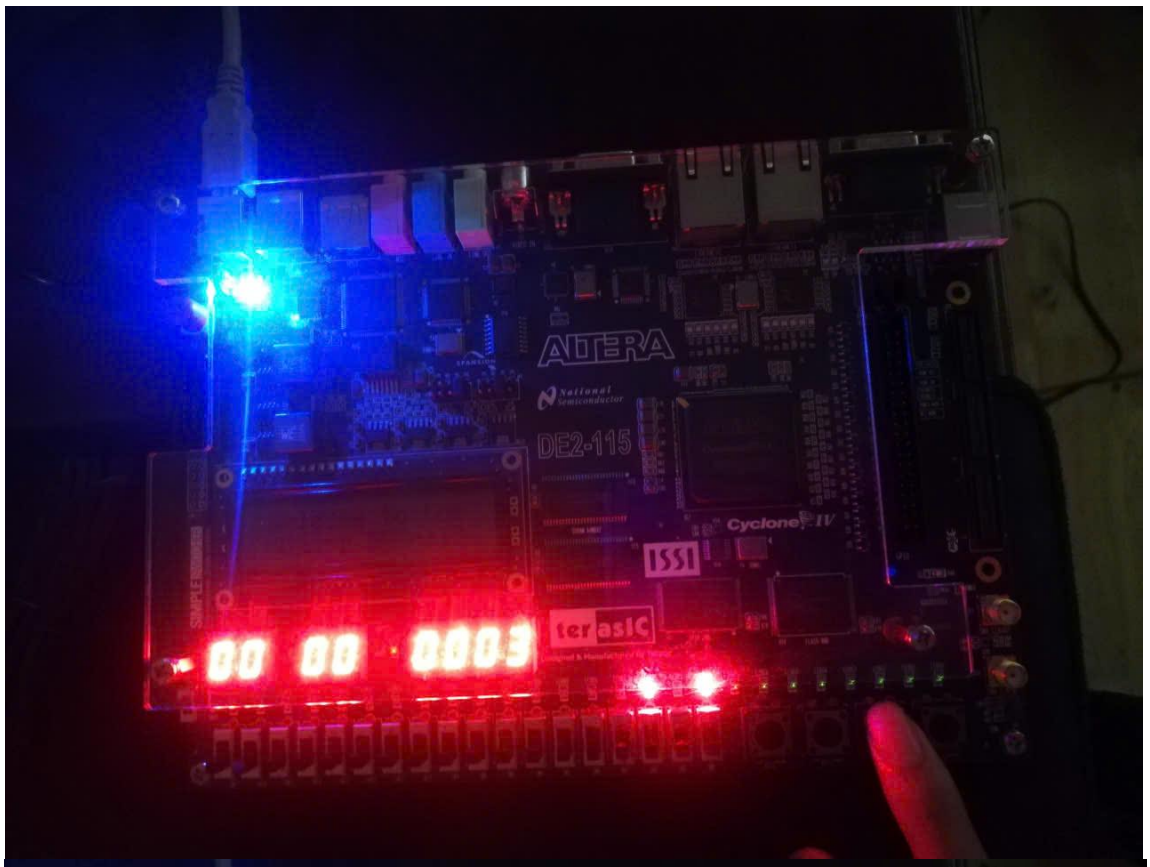

ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
00 00 0002

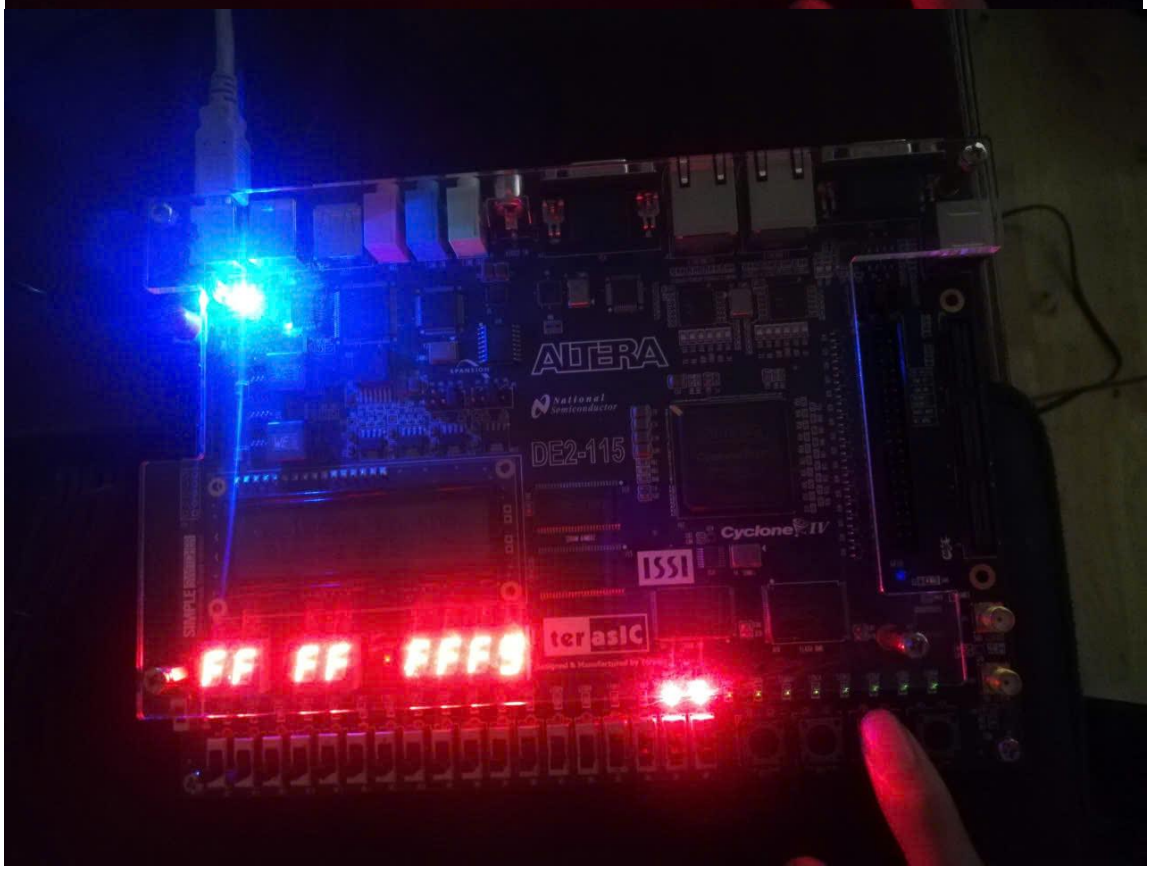

ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
FF FF FFF0

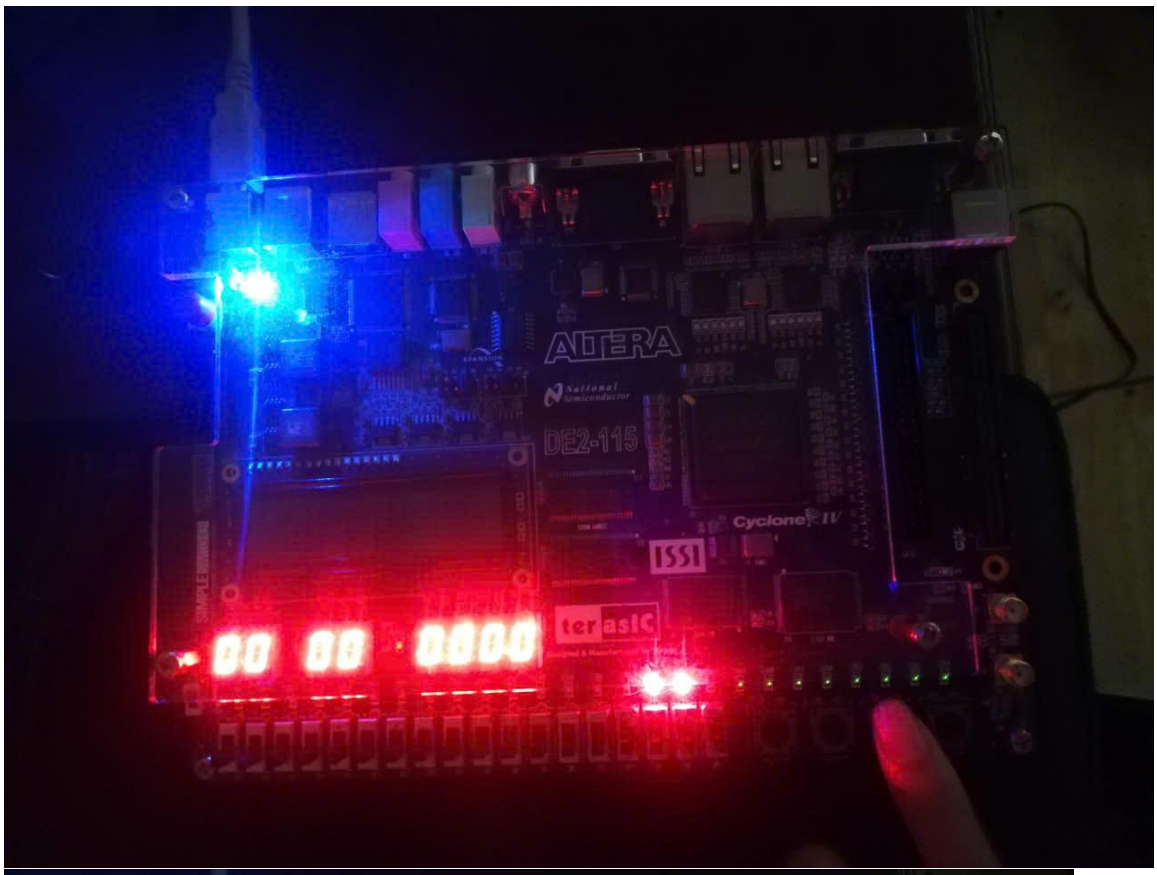

ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
00 00 0000

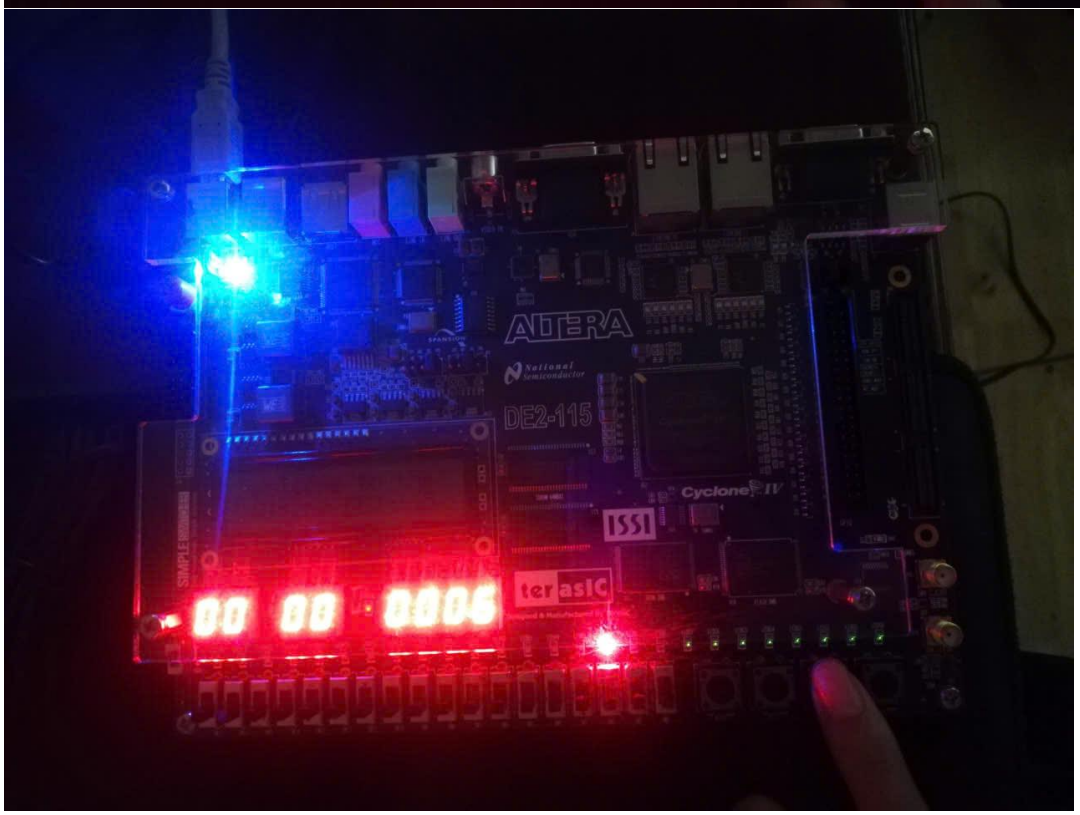

ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
00 00 0003

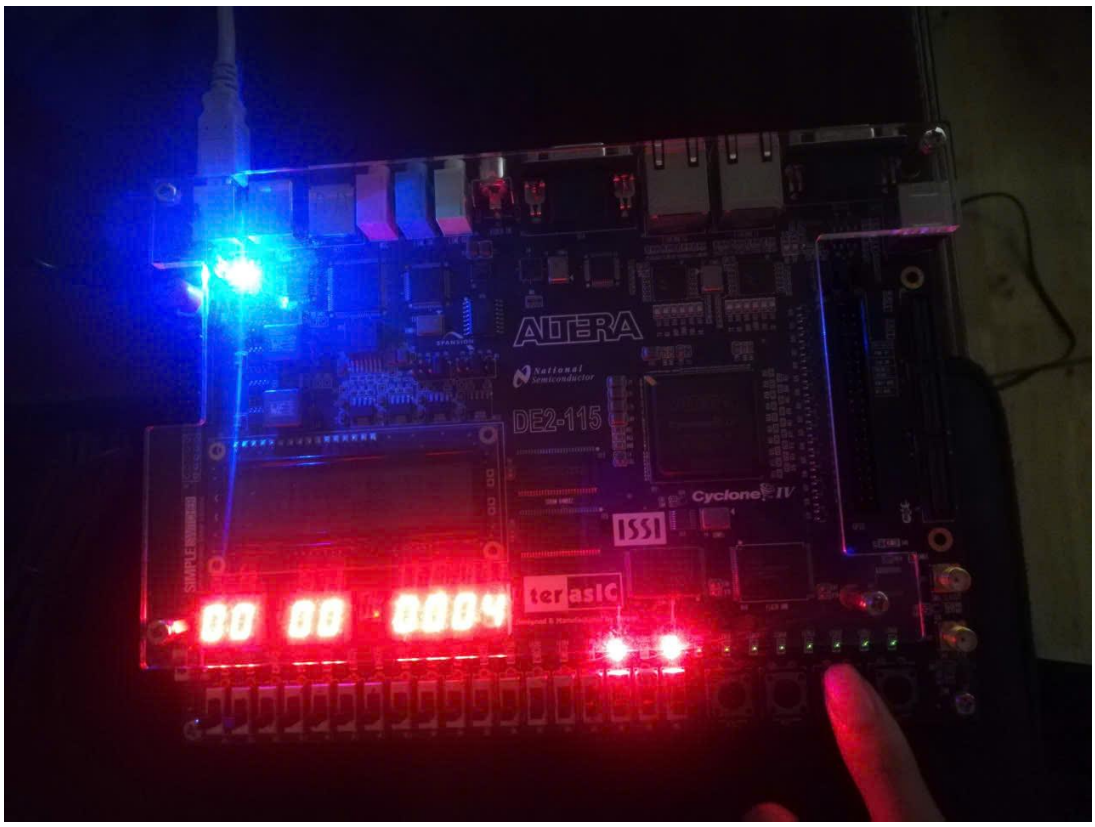

ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
00 00 0000


ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
00 00 0003


ALTERA
DE2-115
Cyclone IV
ISSI
terasIC
FF FF FFF9

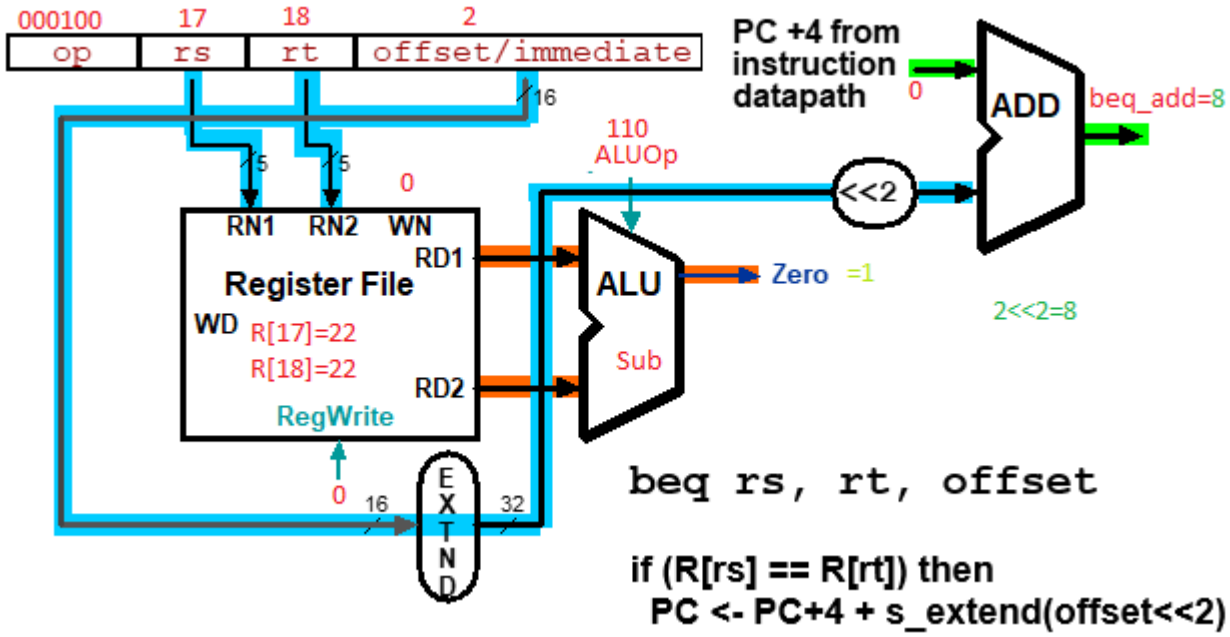

ALTERA
DE2-115
Cyclone IV
ISSI
terasIC


ALTERA
DE2-115
Cyclone IV
ISSI
terasIC



III.5 EXPERIMENT NO. 5

III.5.1 AIM: To implement I-Type Instruction Beq datapath



III.5.2 CODE

module beq_Datapath (rs,rt,offset,ALUOp,PC_plus_4, Zero,beq_add)

endmodule

III.5.3 LAB ASSIGNMENT

1) Write Verilog code to implement beq_Datapath module
2) Write testbenches to verify beq_Datapath module, simulate and verify the output data.

III.10.3 LAB ASSIGNMENT

Write testbenches to verify above blocks and attach waveforms.

III. LAB REPORT GUIDELINES

Students write up a report on the Verilog HDL implementation experiment projects created in this lab. The lab report should include Verilog code for the module under test, Verilog test bench code and a truth table results, and example data input and output to validate the experiment. Simulation Result in form of Simulation Capture Screen.